Министерство образования и науки Российской Федерации Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌ высшего‌ ‌образования‌

**«Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №1**

Дисциплина: «Дискретная математика и математическая логика»

Тема: Калькулятор множеств целых чисел

Выполнила работу

студентка группы ИВТ-22-2б

Ищенко Д. О.

Проверила

Старший преподаватель кафедры ИТАС

Рустамкханова Г. И.

Пермь, 2023

**Постановка задачи**

Необходимо создать «Калькулятор множеств целых чисел», который имеет следующие возможности:

1. Задание универсума (универсум по умолчанию [-10, 10])
2. Задание трёх множеств различными способами:
   1. случайным образом
   2. вручную с клавиатуры
   3. по условию
3. Выполнение операций над множествами:
   1. объединение
   2. пересечение
   3. разность
   4. симметрическая разность
   5. дополнение до универсума
4. Внутренняя проверка на корректность ввода.

**Анализ задачи**

Для реализации множества создан класс CustomSet, имеющий следующие характеристики:

1. Поля:
   1. vector<int> set : хранит элементы множества
   2. int leftUniversum : хранит левую границу универсума для множества
   3. int rightUniversum : хранит правую границу универсума для множества
2. Методы:
   1. Sort() : сортирует вектор элементов множества
   2. CustomSet() : конструктор объекта класса
   3. Define() : перегруженный метод, который задаёт элементы множества
   4. DefineRandom() : метод, который задаёт элементы множества случайным образом в пределах универсума
   5. DefineCondition() : метод, который задаёт элементы множества по условию
   6. Print() : выводит множество в консоль
   7. SetUniversum() : перезадаёт универсум
   8. Intersect() : пересекает два множества
   9. Union() : объединяет два множества
   10. Without() : находит разность двух множеств
   11. Difference() : находит симметрическую разность двух множеств
   12. Complement() : находит дополнение множества
3. Вспомогательные внешние функции:
   1. Partition() и QuickSort() : функции, участвующие в сортировке
   2. FoundBefore() : функция, служащая для корректного заполнения множества
   3. print() : вспомогательная функция печати
   4. Unite() : вспомогательная функция объединения
   5. Intersection() : вспомогательная функция пересечения
   6. without() : вспомогательная функция разности
   7. EnterInt() : функция, проверяющая ввод пользователя на корректность
   8. ReadFile() : считывание множеств по умолчанию из файла
   9. Definition() : функция, объединяющая функции заполнения множеств

Универсум по умолчанию находится в промежутке [-10, 10]. Для перезадания универсума пользователь выбирает соответствующий пункт меню. В случае, если какое-либо из уже существующих множеств выходит за рамки нового универсума, пользователю выводится сообщение об этом и требуется ввести множество заново. Реализован метод SetUniversum().

Множества по умолчанию считываются из файла. Для перезадания множеств пользователь выбирает соответствующий пункт меню. Предусмотрено несколько способов задать множество:

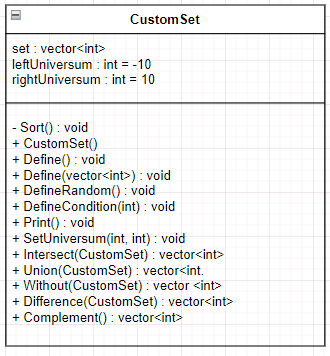
1. Вручную: пользователь вводит количество элементов, а затем сами элементы множества. Реализован метод Define().
2. Случайными числами: пользователь вводит количество элементов, а программа заполняет множество случайными элементами в границах универсума. Реализован метод DefineRandom().
3. По условию: всего доступно несколько условий (чётные числа, нечётные числа, отрицательные числа, числа, кратные n). Пользователь выбирает пункт меню и число n (в случае кратности), программа заполняет множество всеми числами из универсума, подходящими под условие. Реализован метод DefineCondition().

Для работы с множествами предусмотрены пять стандартных операций:

1. Объединение: метод Union() вызывает функцию Unite(), которая обрабатывает два объекта vector<int>. В результирующий vector добавляются все уникальные элементы из двух множеств (уникальность достигается за счёт функции FoundBefore()), затем vector сортируется и возвращается как результат функции.
2. Пересечение: метод Intersect() вызывает функцию Intersection(), которая обрабатывает два объекта vector<int>. В результирующий vector добавляются совпадающие элементы из обоих множеств, с проверкой на уникальность (уникальность достигается за счёт функции FoundBefore()), затем vector сортируется и возвращается как результат функции.
3. Разность: метод Without() вызывает функцию without(), которая обрабатывает два объекта vector<int>. Создаётся временный вектор temp, полученный в результате пересечения двух множеств. В результирующий vector добавляются элементы из первого множества, которых нет в temp, с проверкой на уникальность (уникальность достигается за счёт функции FoundBefore()), за счёт чего результатом будут элементы первого множества без элементов второго множества, затем vector сортируется и возвращается как результат функции.
4. Симметрическая разность: метод Difference() вызывает метод Without() для обоих множеств. Для наглядности результаты выполнения метода для обоих множеств выводятся на экран. Для результатов работы метода вызывается функция Unite(), описанная выше.
5. Дополнение: метод Complement() создаёт вспомогательный vector универсума, который заполняется всеми числами в границах универсума. Далее вызывается функция without(), описанная выше, за счёт чего в результате получится универсум без множества, то есть дополнение множества.

Проверки на корректность ввода и корректность данных реализованы с помощью вспомогательных функций (например, функция EnterInt() проверяет, правильно ли введено целочисленное число), или внутри функций обработки множеств (например, метод SetUniversum() проверяет, есть ли во множестве элементы, которые выходят за новые границы универсума; метод Define() проверяет, правильно ли введён размер множества, который должен быть не больше размера универсума, и проверяет, правильно ли введён элемент, который должен быть уникальным и входить в границы универсума).

**UML-диаграмма класса**



**Тесты**

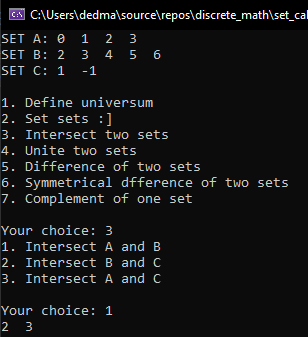


Рисунок 1 - Пересечение множеств A и B

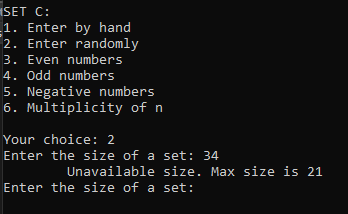


Рисунок 2 - Сообщение об ошибке при неправильно введённом размере множества

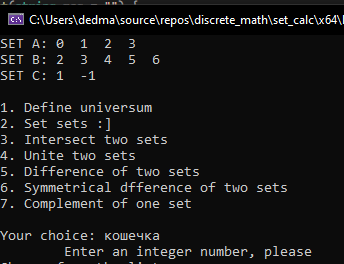


Рисунок 3 - Сообщение об ошибке при вводе слова, а не числа

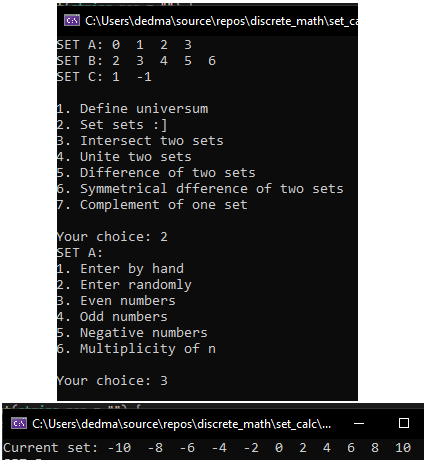


Рисунок 4 - Ввод множества по условию (чётные числа)

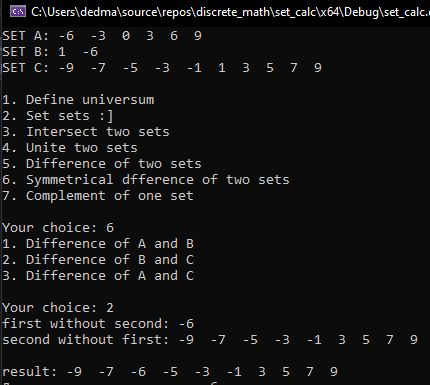


Рисунок 5 - Симметрическая разность двух множеств B и C

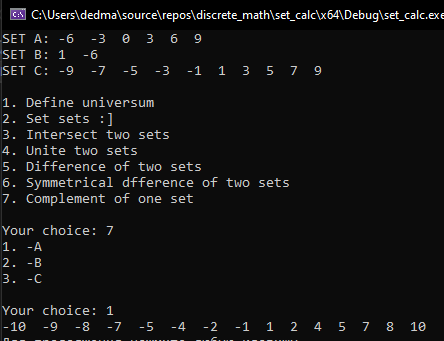


Рисунок 6 - Дополнение ко множеству A

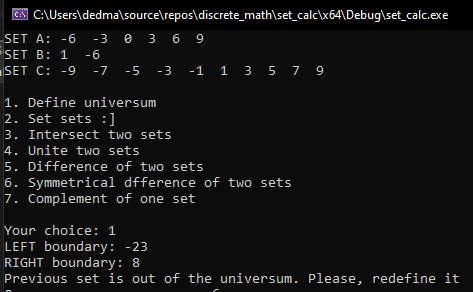


Рисунок 7 - Сообщение о том, что множество выходит за универсум

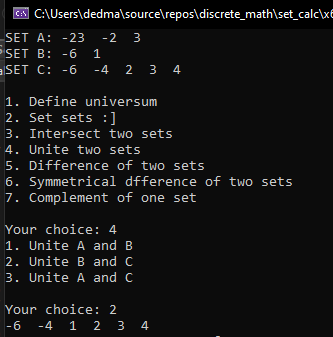


Рисунок 8 - Объединение множеств B и C

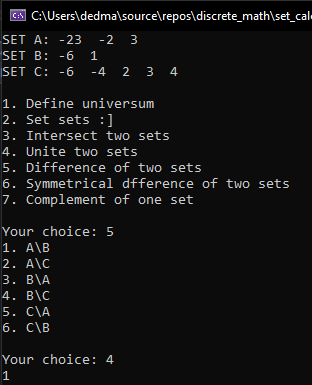


Рисунок 9 - Разность множеств В\С

**Код программы**

Source.cpp

#include <iostream>

#include "CustomSet.h"

using namespace std;

void menu(const int num, CustomSet& a, CustomSet& b, CustomSet& c) {

srand(time(NULL));

switch (num) {

case 1: {

int left = EnterInt("LEFT boundary: ");

int right = EnterInt("RIGHT boundary: ");

a.SetUniversum(left, right);

b.SetUniversum(left, right);

c.SetUniversum(left, right);

break;

}

case 2: {

cout << "SET A:\n";

Definition(&a);

cout << "SET B:\n";

Definition(&b);

cout << "SET C:\n";

Definition(&c);

break;

}

case 3: {

vector<int> res;

cout << "1. Intersect A and B\n";

cout << "2. Intersect B and C\n";

cout << "3. Intersect A and C\n";

int m = EnterInt("\nYour choice: ");

switch (m) {

case 1: {

res = a.Intersect(b);

print(res);

break;

}

case 2: {

res = c.Intersect(b);

print(res);

break;

}

case 3: {

res = a.Intersect(c);

print(res);

break;

}

default: {

cout << "Choose from the list\n";

break;

}

}

break;

}

case 4: {

vector<int> res;

cout << "1. Unite A and B\n";

cout << "2. Unite B and C\n";

cout << "3. Unite A and C\n";

int m = EnterInt("\nYour choice: ");

switch (m) {

case 1: {

res = a.Union(b);

print(res);

break;

}

case 2: {

res = c.Union(b);

print(res);

break;

}

case 3: {

res = a.Union(c);

print(res);

break;

}

default: {

cout << "Choose from the list\n";

break;

}

}

break;

}

case 5: {

vector<int> res;

cout << "1. A\\B\n";

cout << "2. A\\C\n";

cout << "3. B\\A\n";

cout << "4. B\\C\n";

cout << "5. C\\A\n";

cout << "6. C\\B\n";

int m = EnterInt("\nYour choice: ");

switch (m) {

case 1: {

res = a.Without(b);

print(res);

break;

}

case 2: {

res = a.Without(c);

print(res);

break;

}

case 3: {

res = b.Without(a);

print(res);

break;

}

case 4: {

res = b.Without(c);

print(res);

break;

}

case 5: {

res = c.Without(a);

print(res);

break;

}

case 6: {

res = c.Without(b);

print(res);

break;

}

default: {

cout << "Choose from the list\n";

break;

}

}

break;

}

case 6: {

vector<int> res;

cout << "1. Difference of A and B\n";

cout << "2. Difference of B and C\n";

cout << "3. Difference of A and C\n";

int m = EnterInt("\nYour choice: ");

switch (m) {

case 1: {

res = a.Difference(b);

print(res);

break;

}

case 2: {

res = b.Difference(c);

print(res);

break;

}

case 3: {

res = a.Difference(c);

print(res);

break;

}

default: {

cout << "Choose from the list\n";

break;

}

}

break;

}

case 7: {

vector<int> res;

cout << "1. -A\n";

cout << "2. -B\n";

cout << "3. -C\n";

int m = EnterInt("\nYour choice: ");

switch (m) {

case 1: {

res = a.Complement();

print(res);

break;

}

case 2: {

res = b.Complement();

print(res);

break;

}

case 3: {

res = c.Complement();

print(res);

break;

}

default: {

cout << "Choose from the list\n";

break;

}

}

break;

}

default: {

cout << "Choose from the list\n";

break;

}

}

}

int main() {

string path = "myFile.txt";

CustomSet a;

CustomSet b;

CustomSet c;

ReadFile(path, a, b, c);

while (true) {

system("pause");

system("cls");

cout << "SET A: "; a.Print();

cout << "SET B: "; b.Print();

cout << "SET C: "; c.Print();

cout << "\n1. Define universum\n";

cout << "2. Set sets :]\n";

cout << "3. Intersect two sets\n";

cout << "4. Unite two sets\n";

cout << "5. Difference of two sets\n";

cout << "6. Symmetrical dfference of two sets\n";

cout << "7. Complement of one set\n";

int num = EnterInt("\nYour choice: ");

menu(num, a, b, c);

}

return 0;

}

CustomSet.h

#pragma once

#include <vector>

#include <algorithm>

#include <fstream>

using namespace std;

class CustomSet;

int Partition(vector<int>&, int, int);

void QuickSort(vector<int>&, int, int);

bool FoundBefore(vector<int>&, int, int);

void print(vector<int>&);

vector<int> Unite(vector<int>, vector<int>);

vector<int> Intersection(vector<int>, vector<int>);

vector<int> without(vector<int>, vector<int>);

int EnterInt(string);

void ReadFile(string, CustomSet&, CustomSet&, CustomSet&);

void Definition(CustomSet\*);

class CustomSet {

vector<int> set;

int leftUniversum = -10;

int rightUniversum = 10;

void Sort();

public:

CustomSet();

void Define();

void Define(vector<int>);

void DefineRandom();

void DefineCondition(int);

void Print();

void SetUniversum(int, int);

vector<int> Intersect(CustomSet);

vector<int> Union(CustomSet);

vector<int> Without(CustomSet);

vector<int> Difference(CustomSet);

vector<int> Complement();

};

void CustomSet::DefineCondition(int num) {

system("cls");

this->set.clear();

int number, n = numeric\_limits<int>::infinity();

for (int i = leftUniversum; i <= rightUniversum; ++i) {

if (num == 1) { //even

system("cls");

if (abs(i) % 2 == 0) set.push\_back(i);

cout << "Current set: "; print(this->set);

}

if (num == 2) { //odd

system("cls");

if (abs(i) % 2 == 1) set.push\_back(i);

cout << "Current set: "; print(this->set);

}

if (num == 3) { //negative

system("cls");

if (i < 0) set.push\_back(i);

cout << "Current set: "; print(this->set);

}

if (num == 4) {//multiplicity

system("cls");

if (n == numeric\_limits<int>::infinity()) n = EnterInt("Enter n: ");

if (i % n == 0) set.push\_back(i);

cout << "Current set: "; print(this->set);

}

}

}

void CustomSet::DefineRandom() {

srand(time(NULL));

this->set.clear();

int size = EnterInt("Enter the size of a set: ");

while (size > abs(rightUniversum - leftUniversum) + 1) {

if (size != numeric\_limits<int>::infinity()) {

cout << "\tUnavailable size. Max size is " << abs(rightUniversum - leftUniversum) + 1 << "\n";

}

size = EnterInt("Enter the size of a set: ");

system("pause");

system("cls");

}

for (int i = 0; i < size; ++i) {

system("cls");

int number = rand() % abs(rightUniversum - leftUniversum + 1) + leftUniversum;

while (FoundBefore(set, i, number)) {

number = rand() % abs(rightUniversum - leftUniversum + 1) + leftUniversum;

}

set.push\_back(number);

cout << "Current set: "; print(this->set);

}

}

void CustomSet::Define(vector<int> v) {

this->set = v;

}

void CustomSet::Define() {

system("cls");

this->set.clear();

int size, number;

size = EnterInt("Enter the size of a set: ");

while (size > abs(rightUniversum - leftUniversum) + 1) {

if (size != numeric\_limits<int>::infinity()) {

cout << "\tUnavailable size. Max size is " << abs(rightUniversum - leftUniversum) + 1 << "\n";

}

size = EnterInt("Enter the size of a set: ");

system("pause");

system("cls");

}

for (int i = 0; i < size; ++i) {

system("cls");

cout << "Current set: "; print(this->set);

number = EnterInt("Enter element: ");

while (FoundBefore(set, i, number) || (number<leftUniversum) || (number > rightUniversum)) {

cout << "\tPlease, enter a unique element in universum boundaries\n";

number = EnterInt("Enter element: ");

}

set.push\_back(number);

}

}

void CustomSet::SetUniversum(int leftUniversum, int rightUniversum) {

if (leftUniversum > rightUniversum) swap(leftUniversum, rightUniversum) ;

this->leftUniversum = leftUniversum;

this->rightUniversum = rightUniversum;

this->Sort();

if ((this->set[0] < leftUniversum) || (this->set[set.size() - 1] > rightUniversum)) {

cout << "Previous set is out of the universum. Please, redefine it\n";

system("pause");

this->Define();

this->Sort();

}

}

vector<int> CustomSet::Complement() {

vector<int> UNIVERSUM;

for (int i = leftUniversum; i <= rightUniversum; ++i) {

UNIVERSUM.push\_back(i);

}

return without(UNIVERSUM, this->set);

}

vector<int> CustomSet::Difference(CustomSet other) {

vector<int> first = this->Without(other);

vector<int> second = other.Without(\*this);

cout << "first without second: "; print(first);

cout << "second without first: "; print(second);

cout << "\nresult: ";

return Unite(first, second);

}

vector<int> CustomSet::Without(CustomSet other) {

return without(this->set, other.set);

}

vector<int> CustomSet::Intersect(CustomSet other) {

return Intersection(this->set, other.set);

}

vector<int> CustomSet::Union(CustomSet other) {

return Unite(this->set, other.set);

}

CustomSet::CustomSet() {}

void CustomSet::Print() {

print(set);

}

void CustomSet::Sort() {

QuickSort(set, 0, set.size()-1);

}

int Partition(vector<int>& v, int start, int end) {

int pivot = end;

int j = start;

for (int i = start; i < end; ++i) {

if (v[i] < v[pivot]) {

swap(v[i], v[j]);

++j;

}

}

swap(v[j], v[pivot]);

return j;

}

void QuickSort(vector<int>& v, int start, int end) {

if (start < end) {

int p = Partition(v, start, end);

QuickSort(v, start, p - 1);

QuickSort(v, p + 1, end);

}

}

bool FoundBefore(vector<int>& v, int index, int elem) {

if (index == 0) return false;

for (int i = 0; i < index; ++i) {

if (elem == v[i]) return true;

}

return false;

}

void print(vector<int>& v) {

if (v.size() == 0) {

cout << "\nEmpty set\n";

return;

}

for (auto elem : v) {

cout << elem << " ";

}

cout << "\n";

}

vector<int> Unite(vector<int> that, vector<int> other) {

vector<int> result;

for (auto elem : that) {

result.push\_back(elem);

}

for (int j = 0; j < other.size(); ++j) {

if (!FoundBefore(that, that.size(), other[j])) {

result.push\_back(other[j]);

}

}

QuickSort(result, 0, result.size() - 1);

return result;

}

vector<int> Intersection(vector<int> that, vector<int> other) {

vector<int> result;

for (int i = 0; i < that.size(); ++i) {

for (int j = 0; j < other.size(); ++j) {

if ((that[i] == other[j]) && (!FoundBefore(result, result.size(), that[i]))) {

result.push\_back(that[i]);

}

}

}

QuickSort(result, 0, result.size() - 1);

return result;

}

vector<int> without(vector<int> that, vector<int> other) {

vector<int> result, temp = Intersection(that, other);

for (int i = 0; i < that.size(); ++i) {

result.push\_back(that[i]);

for (int j = 0; j < temp.size(); ++j) {

if ((that[i] == temp[j])) {

if (i >= result.size()) {

result.pop\_back();

continue;

}

result.erase(result.begin() + i);

}

}

}

QuickSort(result, 0, result.size() - 1);

return result;

}

int EnterInt(string msg = "") {

cout << msg;

int a;

cin >> a;

if (cin.fail()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "\tEnter an integer number, please\n";

return numeric\_limits<int>::infinity();

}

else return a;

}

void ReadFile(string path, CustomSet& a, CustomSet &b, CustomSet& c) {

ifstream fin;

fin.open(path);

if (!fin.is\_open()) cout << "Error" << endl;

else {

int number, size;

vector<int> array;

fin >> size;

for (int i = 0; i < size; ++i) {

fin >> number;

array.push\_back(number);

}

a.Define(array);

array.clear();

fin >> size;

for (int i = 0; i < size; ++i) {

fin >> number;

array.push\_back(number);

}

b.Define(array);

array.clear();

fin >> size;

for (int i = 0; i < size; ++i) {

fin >> number;

array.push\_back(number);

}

c.Define(array);

array.clear();

cout << "Universum [-10, 10]\n";

cout << "SET A: "; a.Print();

cout << "SET B: "; b.Print();

cout << "SET C: "; c.Print();

}

fin.close();

return;

}

void Definition(CustomSet \*a) {

cout << "1. Enter by hand\n";

cout << "2. Enter randomly\n";

cout << "3. Even numbers\n";

cout << "4. Odd numbers\n";

cout << "5. Negative numbers\n";

cout << "6. Multiplicity of n\n";

int m = EnterInt("\nYour choice: ");

switch (m) {

case 1: {

a->Define();

break;

}

case 2: {

a->DefineRandom();

break;

}

case 3: {

a->DefineCondition(1);

break;

}

case 4: {

a->DefineCondition(2);

break;

}

case 5: {

a->DefineCondition(3);

break;

}

case 6: {

a->DefineCondition(4);

break;

}

default: {

cout << "Choose from list\n";

}

}

}